TEHNIČKO VELEUČILIŠTE U ZAGREBU

POLITEHNIČKI SPECIJALISTIČKI DIPLOMSKI STRUČNI STUDIJ

SPECIJALIZACIJA INFORMATIKA

Stjepan Salopek

RJEŠAVANJE LABIRINATA A* ALGORITMOM PRETRAGE

Zagreb, listopad 2023.

Sadržaj

Uvod

Algoritmi pretrage pokazali su se vrlo korisnima u današnjem svijetu punom informacija u kojem je iznimno bitno pronaći što točniju informaciju, u što kraćem roku, sa što manje resursa.

Postoji mnogo slučajeva gdje se algoritmi pretrage mogu koristiti, a najčešće se radi o pronalasku optimalnog puta, bio on stvarni ili virtualni. Primjerice, često se mogu pronaći u labirintima i društvenim igrama kao što je šah, u obradi prirodnog jezika, usmjeravanju paketa u mreži i tako dalje.

Seminarski rad obraditi će najčešće algoritme pretrage i navesti njihove prednosti i mane. Nadalje, ući će se u određenu dubinu sa A* algoritmom i obrazložiti razlog njegove kvalitete. Na praktičnom primjeru pokazati će se njegova jednostavnost i snaga. Nakon obrade algoritama, uvesti će se u metodu generiranja algoritama u programskom jeziku Python i načina na koji se prikazuju.

Imajući na umu informacije o algoritmima i način rada sa labirintima, pomoću A* algoritma pretrage na jednostavan način će se pronaći najkraći put u labirintu.

Algoritmi pretrage

Algoritam pretrage je skup metoda i procedura namijenjenih istraživanju skupa mogućih rješenja kojeg nazivamo problemskim prostorom [1]. Često se koriste u problemima povezanima sa odabirom optimalnog puta. Dijele se na informirane i neinformirane.

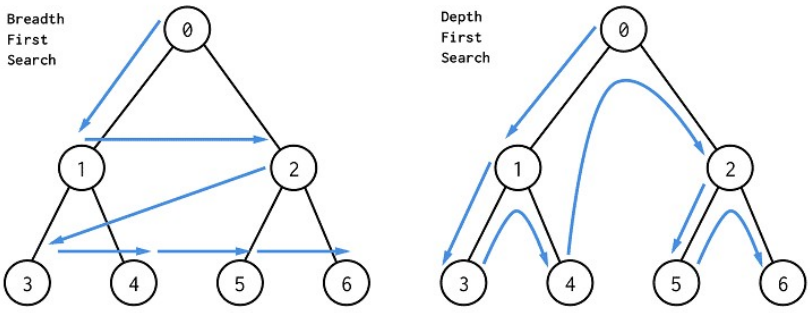
Neinformirani algoritmi imaju informaciju o oznaci cilja do kojeg trebaju doći, no ne znaju njegove koordinate. Informirani algoritmi imaju informaciju o oznaci i poziciji, a zadatak im je pronaći najkraći put do cilja.

BFS i DFS algoritmi pretrage

Breadth-First Search ili pretraživanje u širinu te *Depth-First Search* ili pretraživanje u dubinu su neinformirani algoritmi pretrage. Problemski prostor promatraju kao stablo odluka, u kojemu je korijenski čvor početno stanje algoritma, dok potomci korijena predstavljaju sve moguće opcije i putove kojima se može ići. Ne mora svaki put voditi do cilja.

Algoritam pretraživanja u širinu traži rješenje razinu po razinu, a koristi podatkovnu strukturu reda koji radi na principu *First In, First Out*. U početku svu djecu korijenskog čvora slijedno stavlja u red. Prilikom svake iteracije, djecu prvog člana reda će staviti na kraj te ga nakon obrade izbaciti ukoliko nije pronađen cilj. Na ovaj način postiglo se pretraživanje u širinu.

Pretraživanje u dubinu istražuje jedan po jedan put, a na sljedeći prelazi tek kada dođe do slijepe ulice, odnosno kada određeni čvor nema potomaka. Koristi podatkovnu strukturu stoga koji radi na principu *Last In, First Out*. Uzimajući i obrađujući uvijek posljednjeg dodanog člana, postiže se pretraživanje u dubinu.



Slika 1 Usporedba DFS i BFS algoritama pretraživanja

A* algoritam pretrage

Glavna mana kod DFS i BFS algoritama je njihova neinformiranost. S obzirom da to da nemaju uvid u lokaciju cilja, postoji velika vjerojatnost da će nepotrebno trošiti resurse. Primjerice, prilikom traženja najkraćeg puta od Zagreba do Ljubljane, oba algoritma zanimati će i put koji vodi prema Mađarskoj, iako bi bilo poželjno ignorirati ga.

Da bi se izbjegao ovaj problem, koristi se A* algoritam pretrage koji koristi lokaciju cilja kao nit vodilju za odabir najkraće rute. Na primjeru puta do Slovenije, algoritam bi odabrao i manje kvalitetnu cestu koja vodi prema Sloveniji umjesto kvalitetnije koja vodi prema Mađarskoj, zato jer je preko funkcije određivanja kvalitete uočio da bi ga potonja udaljila od cilja.

Algoritam radi na način da u svakoj iteraciji sagledava neposjećene susjede određenog čvora i na njega primjenjuje funkciju troška koja se za potrebe seminara naziva funkcija *f(čvor)*. Navedena funkcija uzima u obzir stvarnu udaljenost trenutne pozicije do susjednog čvora, naziva *g(čvor)* i heuristiku, odnosno procjenu udaljenosti susjednog čvora do cilja, naziva *h(čvor)*. Matematički izraženo, funkcija se može zapisati na sljedeći način:

$$f(\check{c}vor) = g(\check{c}vor) + h(\check{c}vor)$$

Primjer korištenja algoritma pretraživanja

Kvaliteta funkcije A* algoritma može se uočiti na rješavanju problema pronalaska najkraćeg puta od Zagreba do Splita. Pretpostavimo da je Zagreb početni čvor koji se može granati na Karlovac i Križevce, za koje se zna da su udaljeni od Zagreba pedeset kilometara. Znamo i njihove koordinate na karti. Također, iako ne znamo udaljenost od Splita do Zagreba, znamo i njegove koordinate.

Algoritmi pretrage u širinu i dubinu slijepi su, što znači da ne znaju da nema smisla ići prema Križevcima, zato jer će se time udaljiti od Splita. Oni će vidjeti da su oba grada jednako udaljena, i smatrati će ih jednako kvalitetnima za sljedeći korak na putu do cilja.

S druge strane, A* algoritam će izračunati *f(Karlovac)* i *f(Križevci)*. U postavkama zadatka stoji da *g(Karlovac)* i *g(Križevci)* iznose pedeset kilometara. Pomoću koordinata će izračunati zračne udaljenosti od susjednih čvorova do Splita i saznati da *h(Karlovac)* iznosi dvjesto trideset kilometara, a *h(Križevci)* dvjesto osamdeset.

Rezultati funkcija *f* će iznositi dvjesto osamdeset i tristo trideset kilometara, na čemu će se utvrditi da je Karlovac optimalnija opcija za sljedeći korak u algoritmu.

Rješavanje labirinata

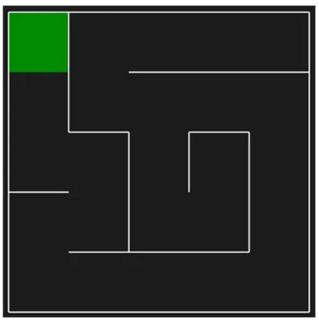
Generiranje i postavljanje labirinta u programskom jeziku Python

Za potrebe rada korišten je članak stranice Medium.com [2]. Labirint se generirao bibliotekom *pyamaze*. Slanjem dimenzija i pokretanjem skripte, labirint se otvara u zasebnom prozoru.

```
from pyamaze import maze

m=maze(5,5)
m.CreateMaze()
m.run()
```

Slika 2 Stvaranje labirinta pyamaze bibliotekom

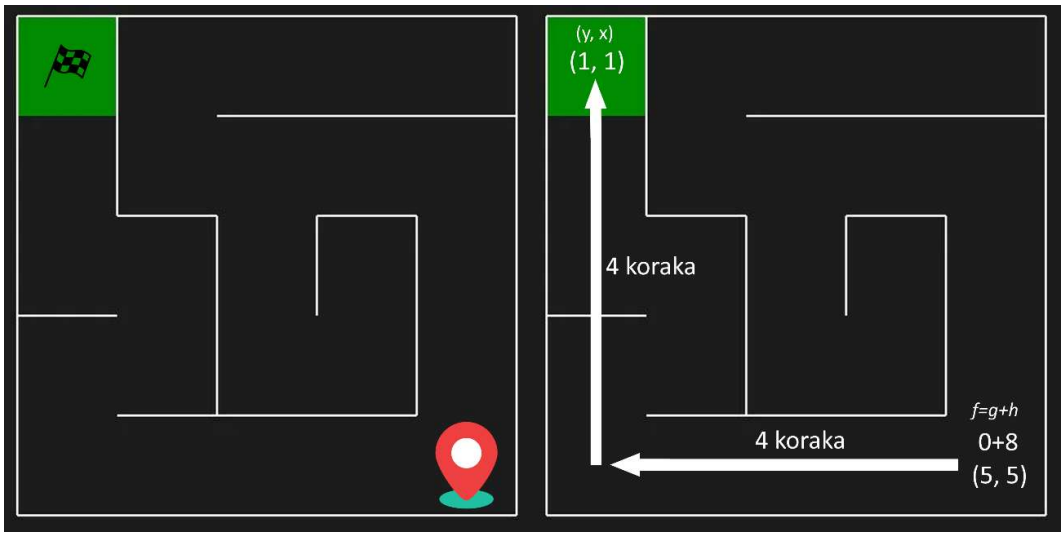


Slika 3 Izgled labirinta

Rješavanje labirinta vrši se na način da se, počevši od ćelije u donjem desnom kutu, iterativno vrši izračun funkcije *f* i odabire susjed sa minimalnom vrijednošću.

Vrijednost funkcije *g* u početnoj točki je nula, a za svaki idući korak uvećava se za jedan. Heuristika ili funkcija *h* se u slučaju labirinata može računati na dva načina. Prvi je uzimati euklidsku udaljenost ćelije do cilja izračunavanjem vrijednosti hipotenuze Pitagorinim poučkom, a druga je izračunavanje takozvane Manhattan vrijednosti, odnosno zbroja krakova pravokutnog trokuta. U seminarskom radu korištena je Manhattan metoda radi jednostavnosti izračuna zbrajanja u odnosu na izračunavanje korijena euklidske metode.

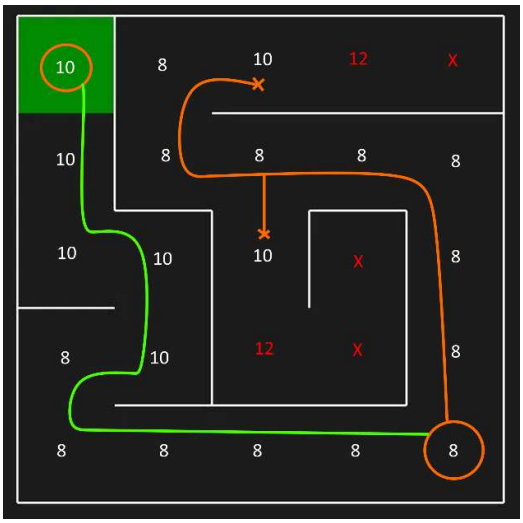
Sljedeća slika prikazuje inicijalno generirani labirint sa početnom ćelijom i ciljem, kao i način na koji se izračunala funkcija troška *f* za početnu ćeliju.



Slika 4 Postavljanje labirinta

Rješavanje labirinta

Imajući postavljenu početnu ćeliju, kreće se u iterativni izračun funkcije troška sve do pronalaska cilja. Može se primijetiti da mnoge ćelije nakon početne imaju vrijednost funkcije troška *f* u iznosu osam. Razlog tome je taj da se svakim korakom vrijednost funkcije *g* povećavala za jedan, a Manhattan udaljenost od cilja funkcije *h* za istu vrijednost smanjivala.



Slika 5 Rješavanje labirinta A* metodom

U slučaju kada se susjedna ćelija krenula udaljavati od cilja, funkcija *h* se povećala za jedan, a vrijednost troška ćelije prešla je na 10.

U takvim situacijama traži se drugi susjed sa manjom vrijednošću troška. Kada je cijeli labirint mapiran na taj način, može se krenuti u potragu.

Traženjem minimalne vrijednosti troška, algoritam je od početne ćelije krenuo sjeverno u potrazi za ciljem. Dolaskom do vrijednosti troška dvanaest odustao je od daljnje potrage i krenuo zapadno od početne ćelije, čime je u konačnici došao do cilja. Crvene ćelije nisu bile posjećene, čime se uštedilo na računalnim resursima.

Zaključak

Na temelju svih informacija navedenih u seminarskom radu, mogu se vidjeti glavne značajke čestih algoritama pretraživanja, a to su pretraživanje u dubinu, u širinu te A* algoritam pretraživanja. Na praktičnom primjeru traženja najkraće rute od Zagreba do Splita mogle su se uvidjeti karakteristike A* algoritma, a to su jednostavnost implementacije i optimizacije potrošnje resursa.

Nadalje, obrađena je tema kreiranja i rješavanja labirinata u programskom jeziku Python. Prikazano je kako se pomoću biblioteke *pyamaze* stvara labirint te je slikovito objašnjeno na koji način ga algoritam mapira i naposljetku rješava.

Popis slika

Tablica izvora

[1] [Mrežno]. Available: https://skolakoda.github.io/algoritmi-pretrazivanja.

[2] »Medium.com,« [Mrežno]. Available: https://levelup.gitconnected.com/a-star-a-search-for-solving-a-maze-using-python-with-visualization-b0cae1c3ba92.